Boosting Algorithms

ADABOOST

For example, if we were using this data to determine if someone had heart disease or not…

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Many classifiers like LR, Decision trees, Naive Bayes, SVM NN, Random Forest (Collection of Decision Trees) or **Adaboost** can be used.

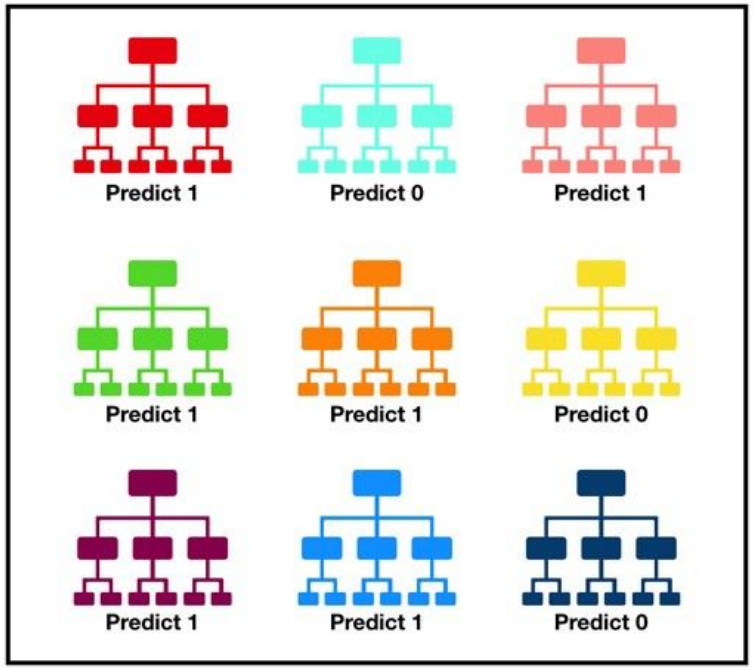
Bagging v/s Boosting

1. Bagging only **controls high variance** in a model while boosting controls both bias and variance. So, boosting is considered to be more effective.

2. In bagging, each weak learner has **equal say** in the final decision while in boosting the weak learner which generates high accuracy has more say in the final decision.

3. In bagging, all **weak learners are independent** of each other. The **sequence** in which each weak learner is created does not matter. But in boosting, the sequence of creation of each weak learner does matter.

Random forest is a **bagging** technique and not a **boosting** technique. The trees in **random forests** are run in parallel. There is no interaction between these trees while building the trees.



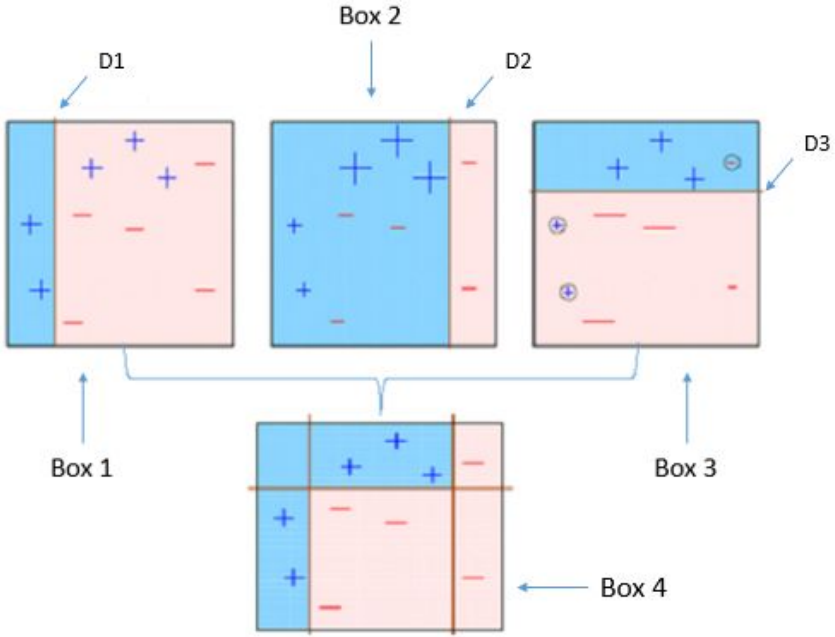
Tally: Six 1s and Three 0s

Prediction: 1

Notice that with bagging we are not subsetting the training data into smaller chunks and training each tree on a different chunk. Rather, if we have a sample of size N, we are still feeding each tree a training set of size N (unless specified otherwise). But instead of the original training data, we take a random sample of size N with replacement. For example, if our training data was [1, 2, 3, 4, 5, 6] then we might give one of our trees the following list [1, 2, 2, 3, 6, 6]. Notice that both lists are of length six and that “2” and “6” are both repeated in the randomly selected training data we give to our tree (because we sample with replacement).

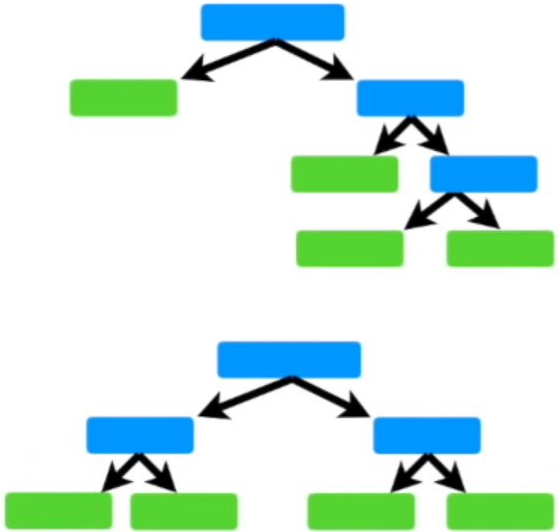
Decisions trees are very sensitive to the data they are trained on — small changes to the training set can result in significantly different tree structures. Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is known as **bagging**.

The term '**Boosting**' refers to a family of **algorithms** which converts weak learner to strong learners. **Boosting** is an ensemble method for improving the model predictions of any given learning **algorithm**. The idea of **boosting** is to train weak learners sequentially, each trying to correct its predecessor.



So let's start by using **Decision Trees** and **Random Forests** to explain the three main concepts behind **AdaBoost**!

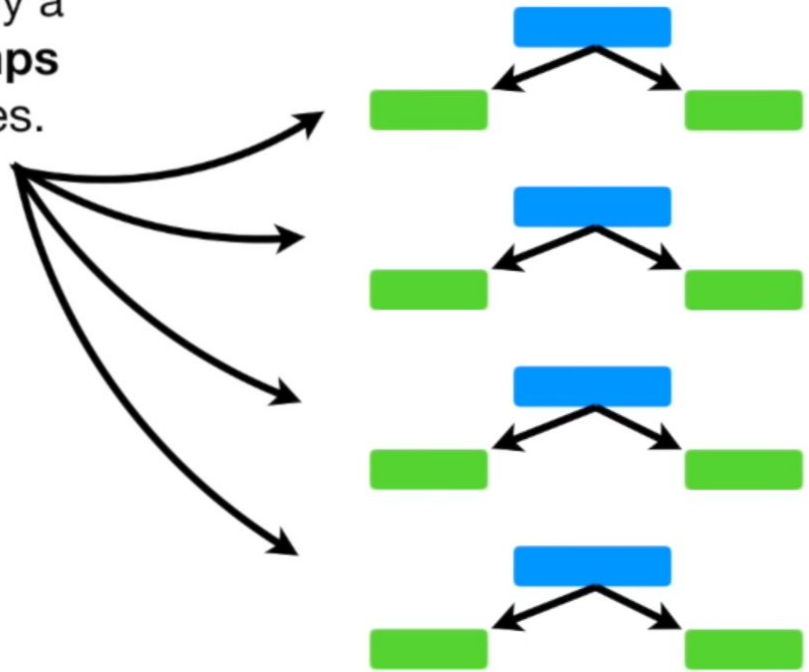
In a **Random Forest**, each time you make a tree, you make a full sized tree.



Some trees might be bigger than others, but there is no predetermined maximum depth.

In contrast, in a **Forest of Trees** made with **AdaBoost**, the trees are usually just a **node** and two **leaves**.

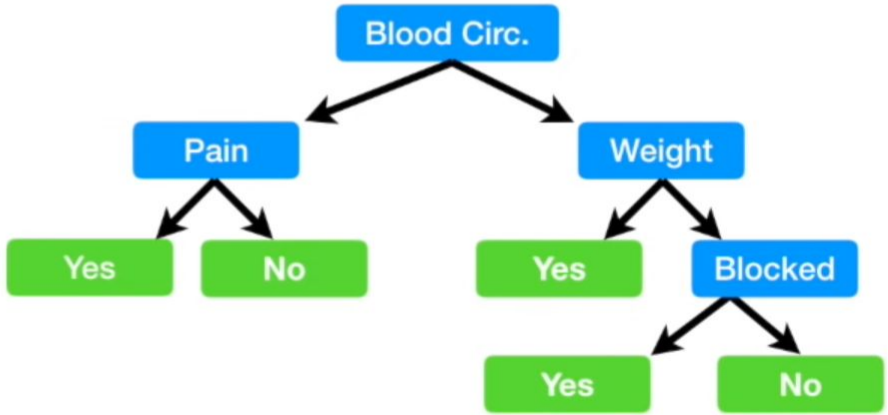
…so this is really a **Forest of Stumps** rather than trees.



For example, if we were using this data to determine if someone had heart disease or not…

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

…then a full sized **Decision Tree** would take advantage of all **4** variables that we measured (**Chest Pain**, **Blood Circulation**, **Blocked Arteries** and **Weight**) to make a decision…



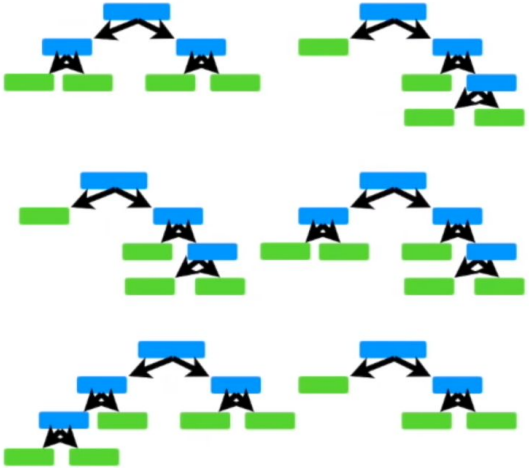
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

…but a **Stump** can only use one variable to make a decision.

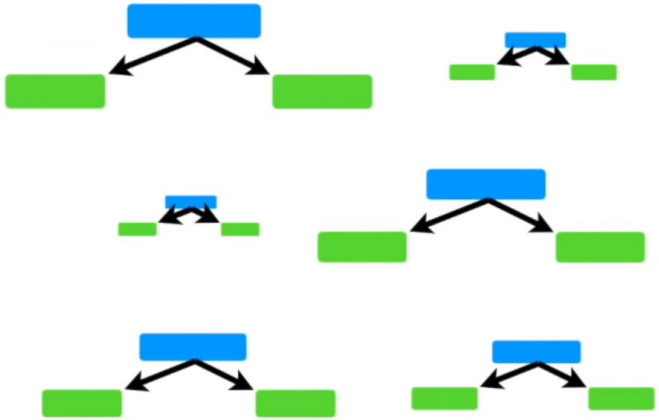
Thus, **Stumps** are technically "weak learners".

These weak learners have **high bias and low variance**.

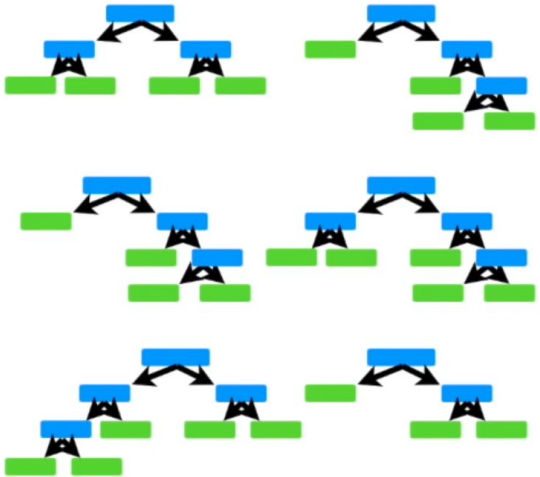
In a **Random Forest**, each tree has an equal vote on the final classification.



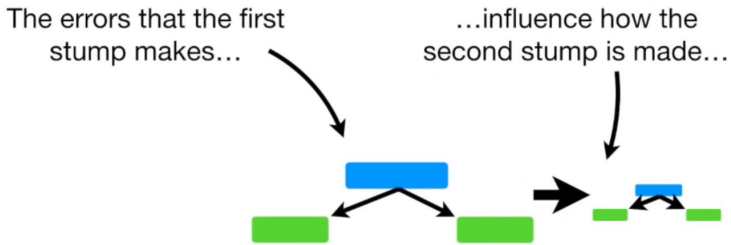
In contrast, in a **Forest of Stumps** made with **AdaBoost**, some stumps get more say in the final classification than others.



Lastly, in a **Random Forest**, each decision tree is made independently of the others.

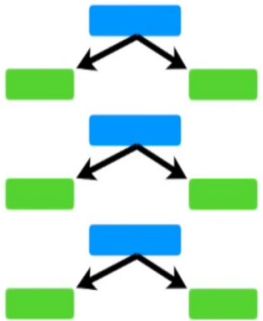


In contrast, in a **Forest of Stumps** made with **AdaBoost**, order is important.

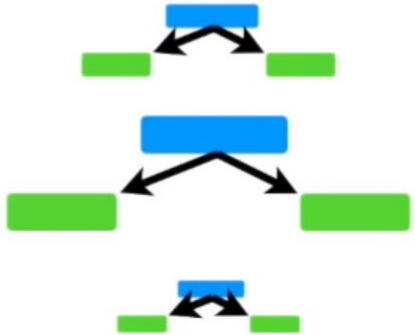


Three ideas behind Adaboost.

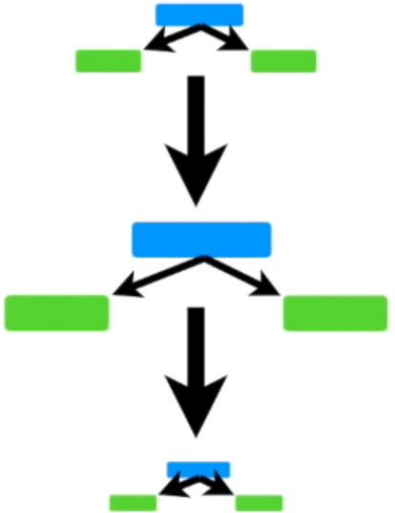
1) **AdaBoost** combines a lot of "weak learners" to make classifications. The weak learners are almost aways **stumps**.



2) Some **stumps** get more say in the classification than others.



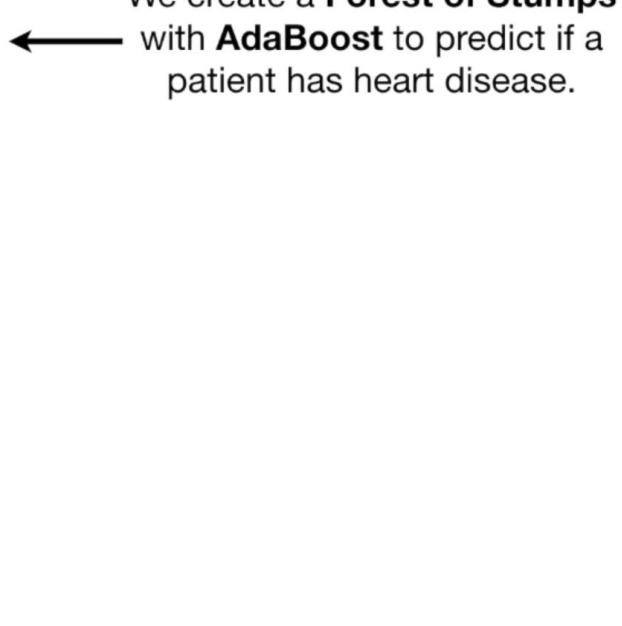
3) Each **stump** is made by taking the previous **stump's** mistakes into account.



Random Forest aims to decrease variance not bias while Adaboost aims to decrease bias not variance.

Now let's dive into the nitty gritty detail of how to create a **Forest of Stumps** using **AdaBoost**.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
Yes	Yes	205	Yes
No	Yes	180	Yes
Yes	No	210	Yes
Yes	Yes	167	Yes
No	Yes	156	No
No	Yes	125	No
Yes	No	168	No
Yes	Yes	172	No



We create a **Forest of Stumps** with **AdaBoost** to predict if a patient has heart disease.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	
No	Yes	180	Yes	
Yes	No	210	Yes	
Yes	Yes	167	Yes	
No	Yes	156	No	
No	Yes	125	No	
Yes	No	168	No	
Yes	Yes	172	No	

The first thing we do is give each sample a weight that indicates how important it is to be correctly classified.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

At the start, all samples get the same weight…

$$\frac{1}{\text{total number of samples}} = \frac{1}{8}$$

…and that makes the samples all equally important.

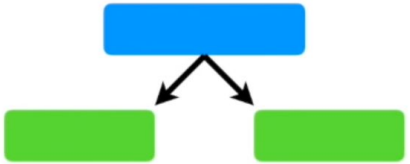
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

However, after we make the first stump, these weights will change in order to guide how the next stump is created.

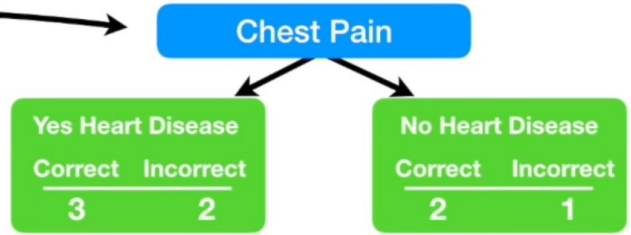
How to make the first stump.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

NOTE: Because all of the weights are the same, we can ignore them right now.

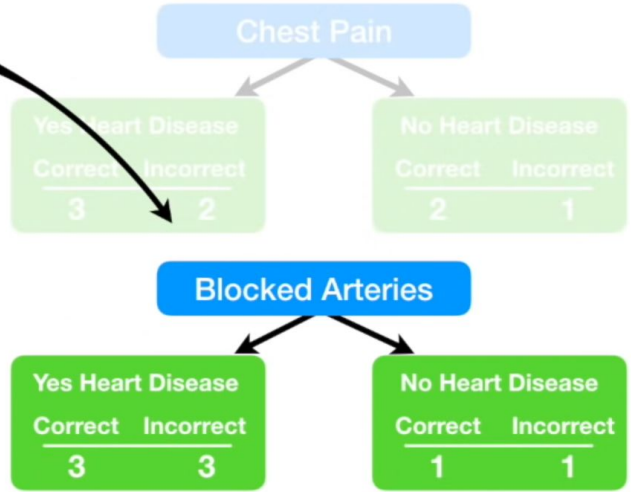


Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8



We start by seeing how well **Chest Pain** classifies the samples.

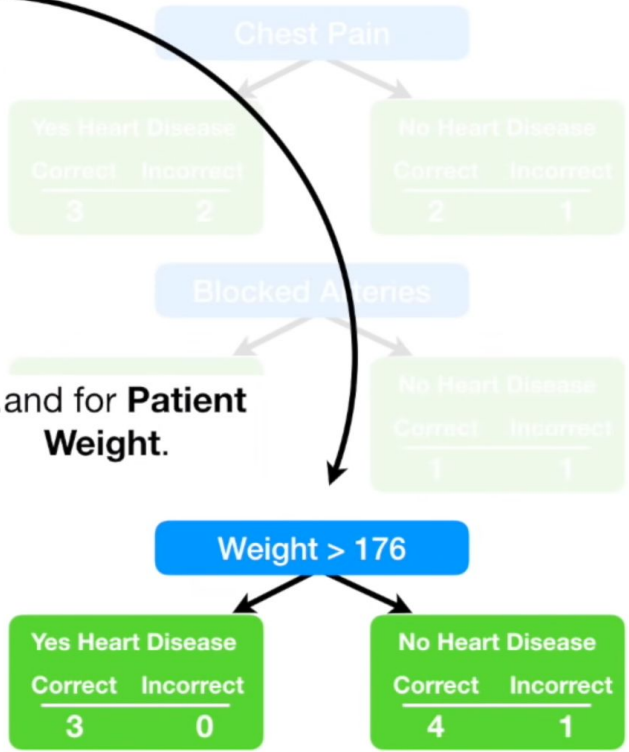
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

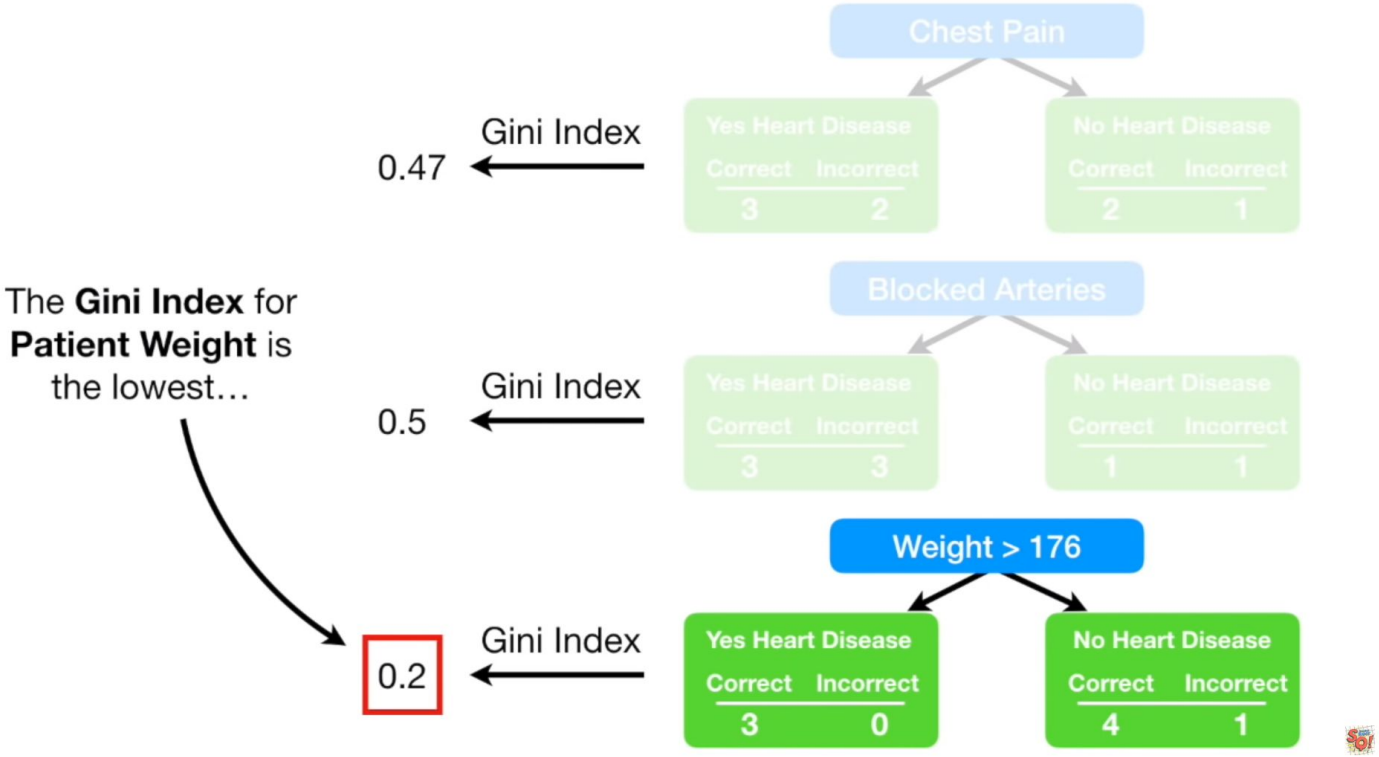


Now we do the same thing for **Blocked Arteries**…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

…and for **Patient Weight**.




Chest Pain
Yes Heart Disease
Correct Incorrect
3 2
No Heart Disease
Correct Incorrect
2 1
Gini Index
0.47
Blocked Arteries
Yes Heart Disease
Correct Incorrect
3 3
No Heart Disease
Correct Incorrect
1 1
Gini Index
0.5
The Gini Index for Patient Weight is the lowest…
Weight > 176
Yes Heart Disease
Correct Incorrect
3 0
No Heart Disease
Correct Incorrect
4 1
Gini Index
0.2

Now we need to determine how much say this stump will have in the final classification.

…so this will be the first stump in the forest.

We determine how much say a stump has in the final classification based on how well it classified the samples.

The Gini Index is determined by deducting the sum of squared of probabilities of each class from one, mathematically, Gini Index can be expressed as:

$$\text{Gini Index} = 1 - \sum_{i=1}^{n} (P_i)^2$$

Where Pi denotes the probability of an element being classified for a distinct class.
where 0 expresses the purity of classification, i.e. All the elements belong to a specified class or only one class exists there. And 1 indicates the random distribution of elements across various classes. The value of 0.5 of the Gini Index shows an equal distribution of elements over some classes, **1** denotes that the elements are randomly distributed across various classes..
Gini index and entropy is the criterion for calculating information gain. Both **gini** and **entropy** are measures of impurity of a node. A node having multiple classes is impure whereas a node having only one class is pure. **Entropy** in statistics is analogous to **entropy** in thermodynamics where it signifies disorder.

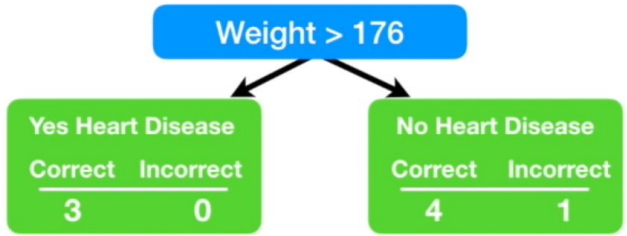
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

In this case, total error is ⅛.

The **Total Error** for a stump is the sum of the weights associated with the *incorrectly* classified samples.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

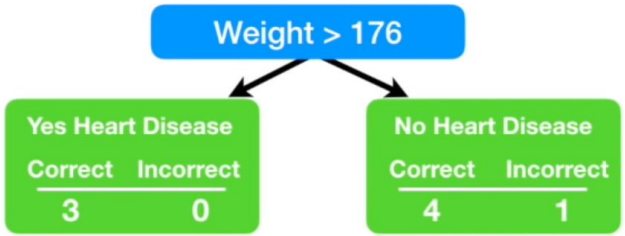
NOTE: Because all of the **Sample Weights** add up to **1**, **Total Error** will always be between **0**, for a perfect stump, and **1**, for a horrible stump.

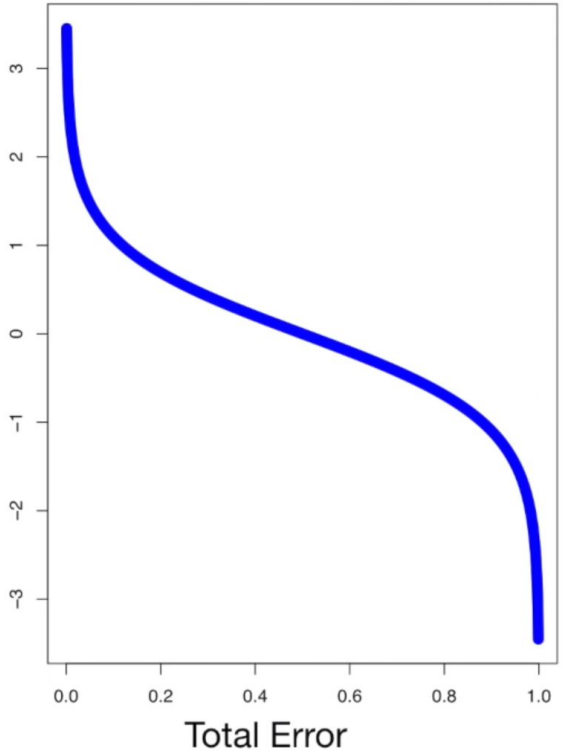


Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

We use the **Total Error** to determine **Amount of Say** this stump has in the final classification with the following formula:

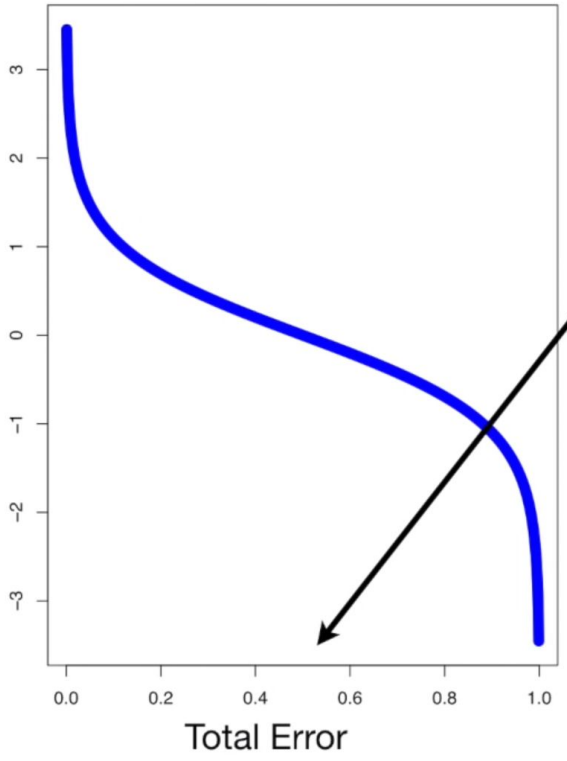
$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$





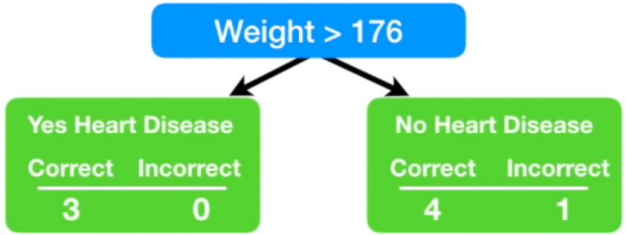
We can draw a graph of the **Amount of Say** by plugging in a bunch of numbers between **0** and **1** for **Total Error**.

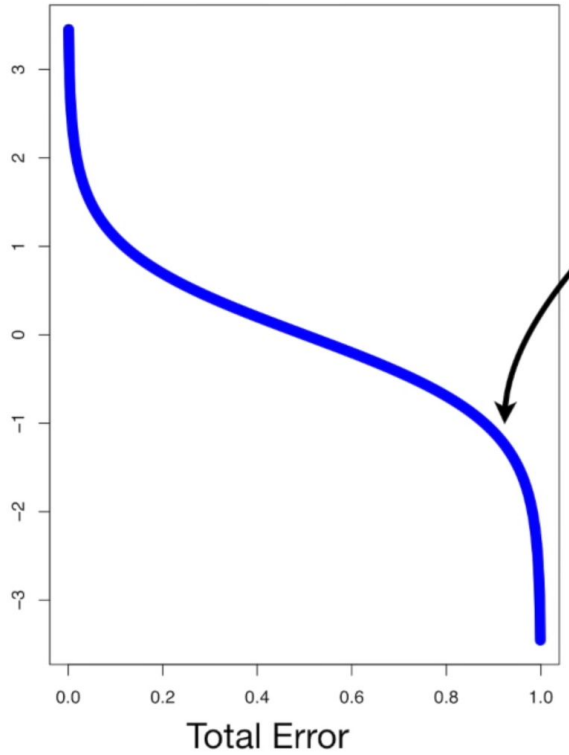
$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$



When a stump is no better at classification than flipping a coin (i.e. half of the samples are correctly classified and half are incorrectly classified) and **Total Error** = **0.5**…

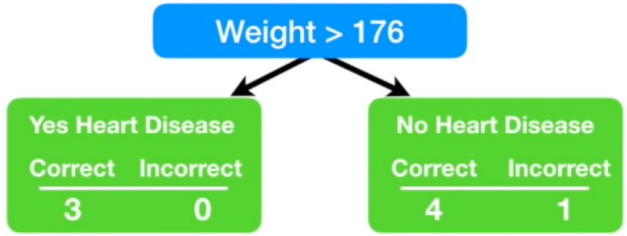
$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$



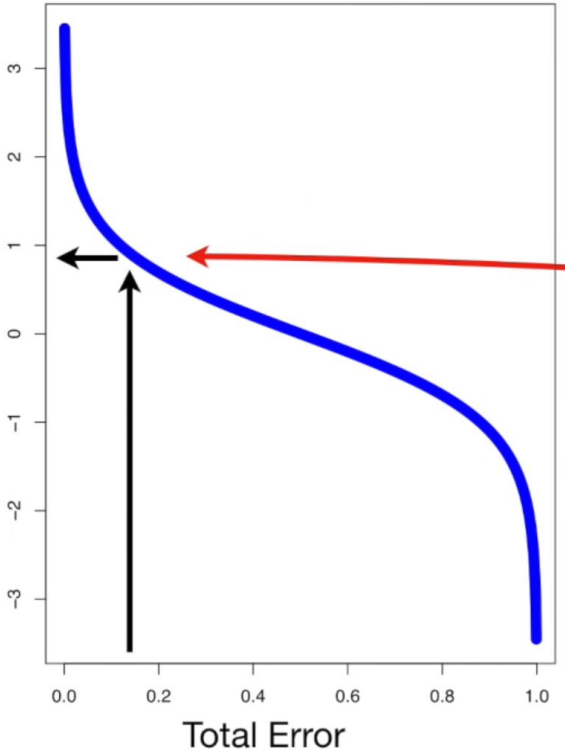


So if a stump votes for "Heart Disease", the negative **Amount of Say** will turn that vote into "*Not* Heart Disease".

$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$



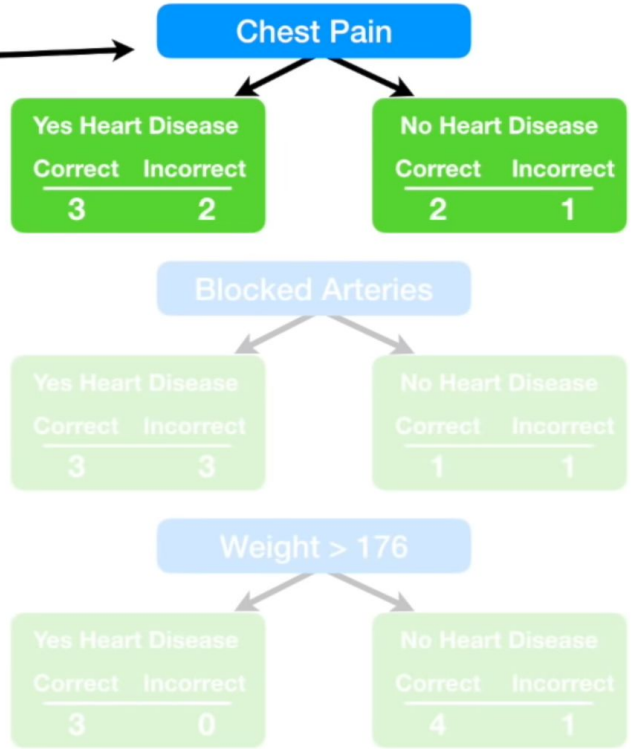
As the Total error for the stump is ⅛.

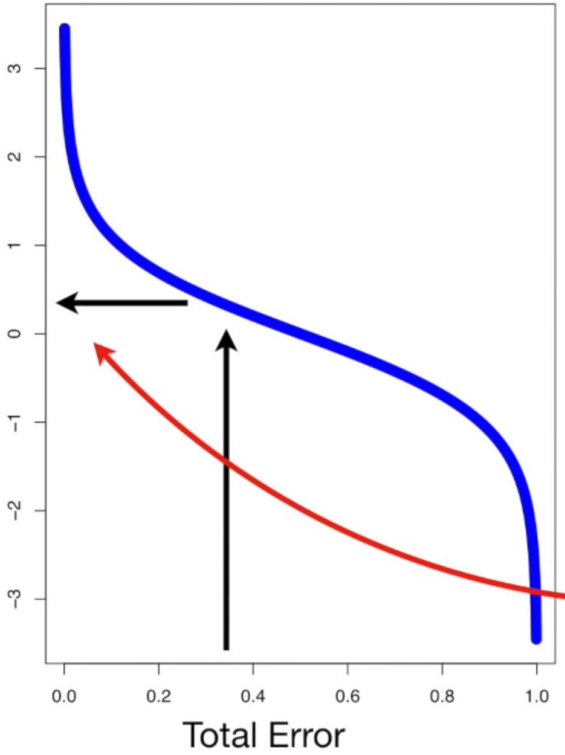


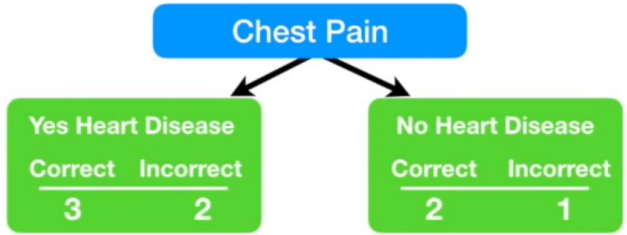
…and the **Amount of Say** that this stump has on the final classification is **0.97**.

$$\text{Amount of Say} = \frac{1}{2}\log(7) = 0.97$$

…let's work out how much say the **Chest Pain** stump would have had if it had been the best stump.

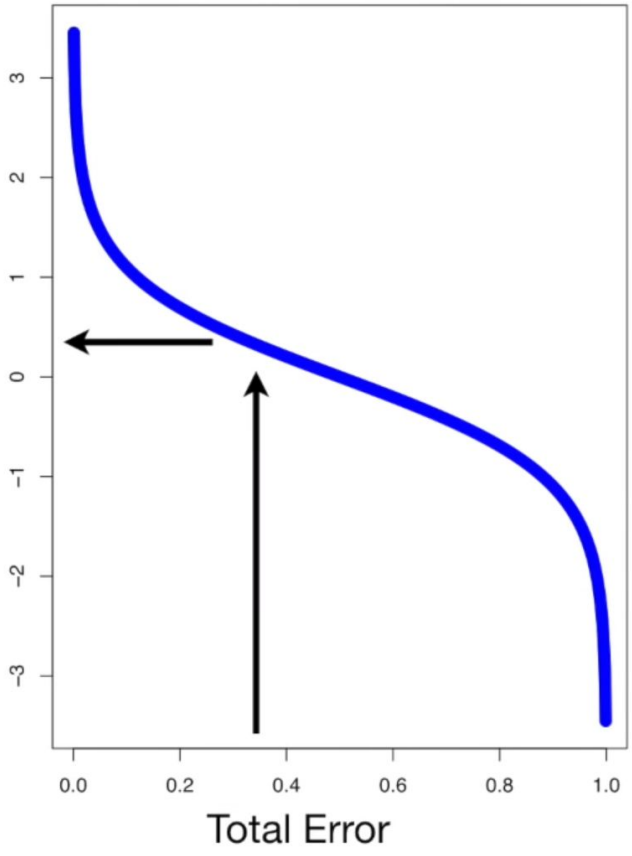


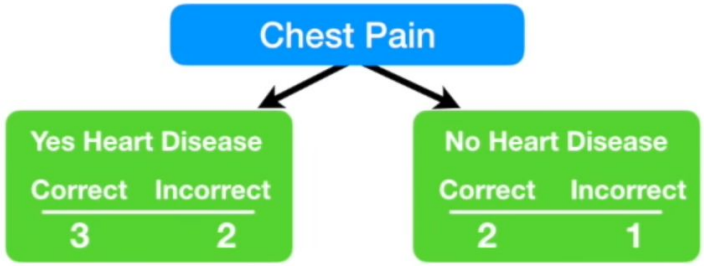




$$\text{Total Error} = \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = \boxed{\frac{3}{8}}$$

So we are expecting the **Amount of Say** to be between 0 and 0.5.





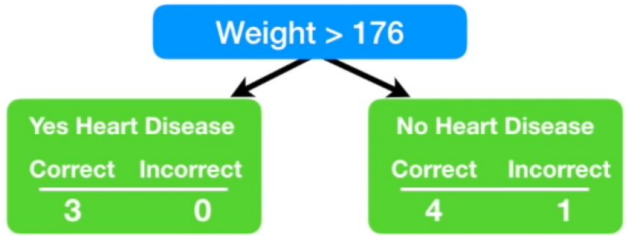
Now we plug **3/8** into the formula for the **Amount of Say** and do the math…

$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - 3/8}{3/8}\right)$$

Now we need to learn how to modify the weights so that the next stump will take the errors that the current stump made into account.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

…and that meant we did not emphasize the importance of correctly classifying any particular sample…



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

…we will emphasize the need for the next stump to correctly classify it by increasing its **Sample Weight**…

…and decreasing all of the other **Sample Weights**.

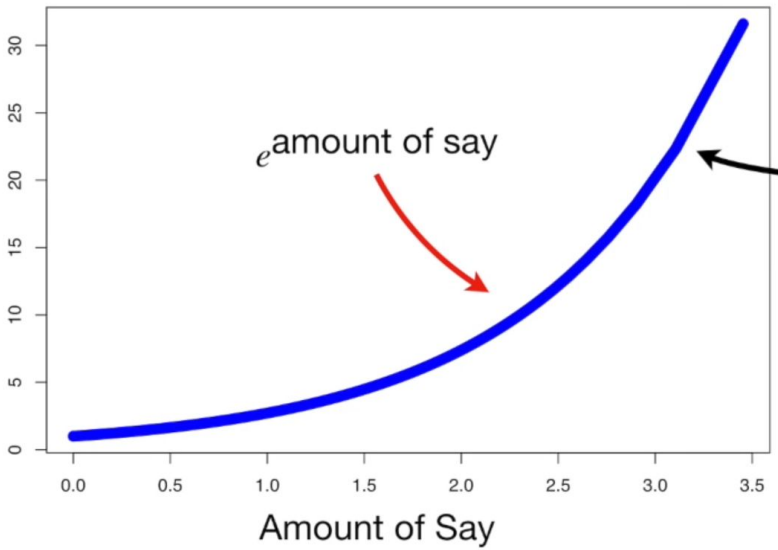
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

This is the formula we will use to *increase* the **Sample Weight** for the sample that was *incorrectly* classified.

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

$$= \frac{1}{8} e^{\text{amount of say}}$$

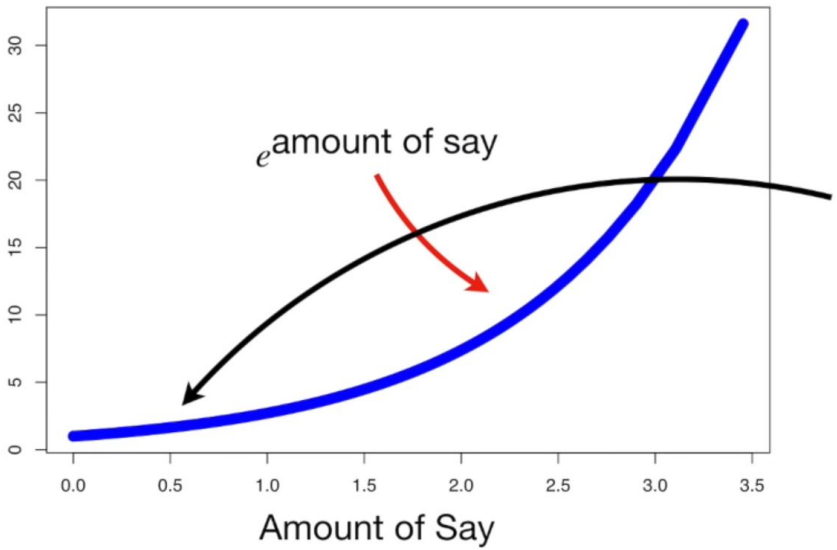


…then we will scale the previous **Sample Weight** with a large number.

This means that the **New Sample Weight** will be much larger than the old one.

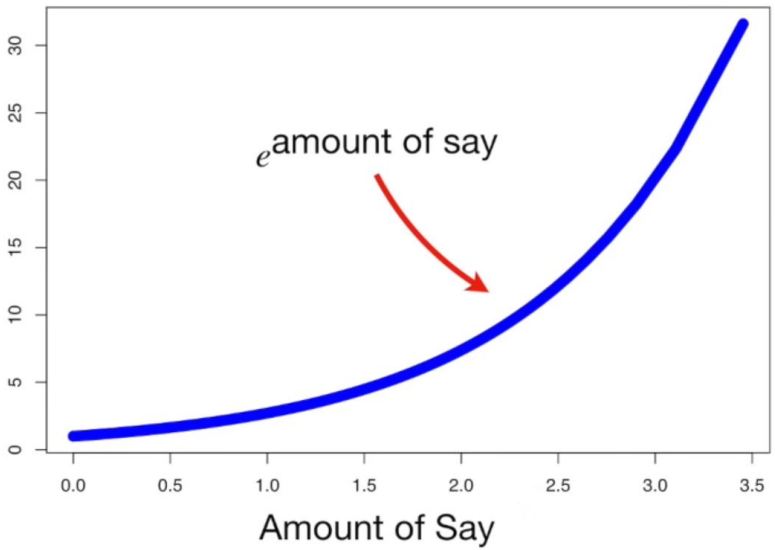
$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

$$= \frac{1}{8} e^{\text{amount of say}}$$



…then the previous **Sample Weight** is scaled by a relatively small number.

This means that the **New Sample Weight** will only be a little larger than the old one.



$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

$$= \frac{1}{8} e^{\text{amount of say}}$$

$$= \frac{1}{8} e^{0.97} = \frac{1}{8} \times 2.64 = 0.33$$

That means the new **Sample Weight** is **0.33**, which is *more* than the old one (**1/8** = **0.125**).

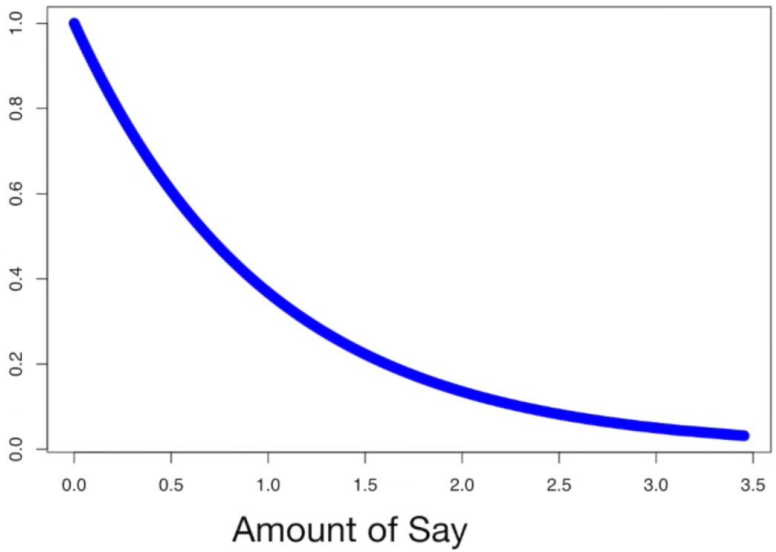
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

Now we need to *decrease* the **Sample Weights** for all of the *correctly* classified samples.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{-amount of say}}$$

This is the formula we will use to *decrease* the **Sample Weights**.



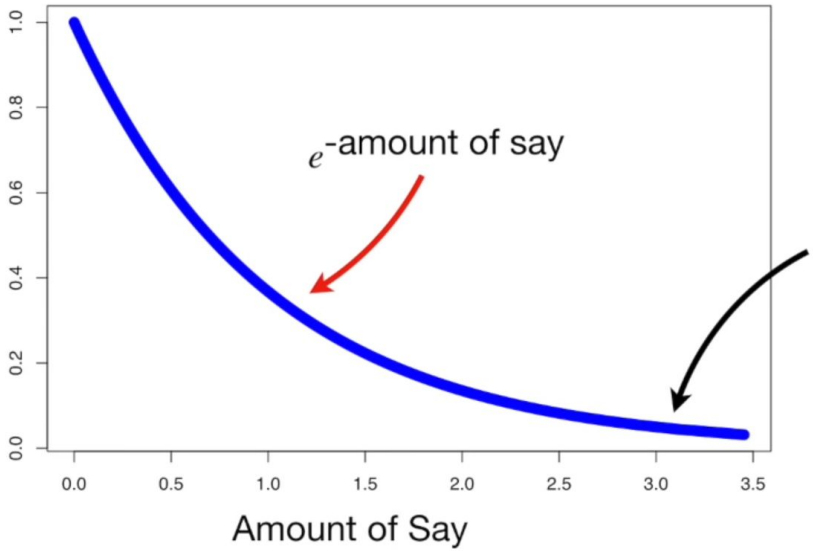
$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{-amount of say}}$$

$$= \frac{1}{8} e^{\text{-amount of say}}$$

…and just like before, we can get a better understanding of how this will scale the **Sample Weight** by plotting a graph using different values for **Amount of Say**.

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{-amount of say}}$$

$$= \frac{1}{8} e^{\text{-amount of say}}$$

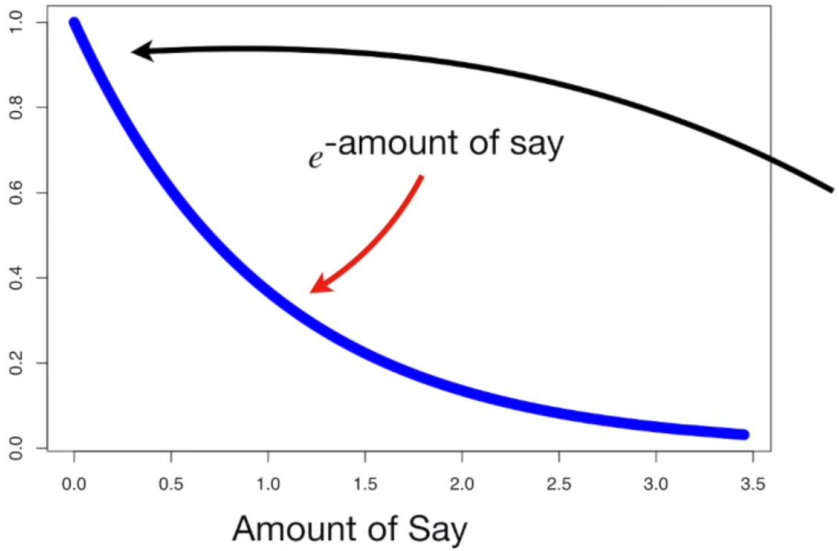


…then we scale the **Sample Weight** by a value very close to 0.

This will make the **New Sample Weight** very small.

New Sample Weight = sample weight $\times e^{\text{-amount of say}}$

$= \frac{1}{8} e^{\text{-amount of say}}$



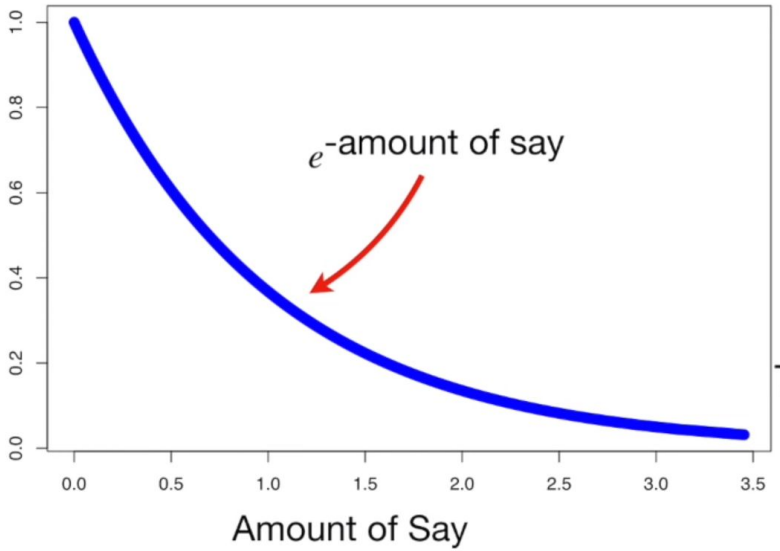
…then we will scale the **Sample Weight** by a value close to **1**.

This means that the **New Sample Weight** will be just a little smaller than the old one.

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{-amount of say}}$$

$$= \frac{1}{8} e^{\text{-amount of say}}$$

$$= \frac{1}{8} e^{-0.97} = \frac{1}{8} \times 0.38 = 0.05$$



The new **Sample Weight** is **0.05**, which is *less* than the old one (**1/8** = **0.125**).

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	
No	Yes	180	Yes	1/8	
Yes	No	210	Yes	1/8	
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	
No	Yes	125	No	1/8	
Yes	No	168	No	1/8	
Yes	Yes	172	No	1/8	

We plug in **0.33** for the sample that was *incorrectly* classified…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	0.05
No	Yes	180	Yes	1/8	0.05
Yes	No	210	Yes	1/8	0.05
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	0.05
No	Yes	125	No	1/8	0.05
Yes	No	168	No	1/8	0.05
Yes	Yes	172	No	1/8	0.05

All of the other samples get **0.05**.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	0.05
No	Yes	180	Yes	1/8	0.05
Yes	No	210	Yes	1/8	0.05
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	0.05
No	Yes	125	No	1/8	0.05
Yes	No	168	No	1/8	0.05
Yes	Yes	172	No	1/8	0.05

Now we need to normalize the **New Sample Weights** so that they will add up to 1.

Right now, if you add up the **New Sample Weights**, you get **0.68**.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight	Norm. Weight
Yes	Yes	205	Yes	1/8	0.05	0.07
No	Yes	180	Yes	1/8	0.05	0.07
Yes	No	210	Yes	1/8	0.05	0.07
Yes	Yes	167	Yes	1/8	0.33	0.49
No	Yes	156	No	1/8	0.05	0.07
No	Yes	125	No	1/8	0.05	0.07
Yes	No	168	No	1/8	0.05	0.07
Yes	Yes	172	No	1/8	0.05	0.07

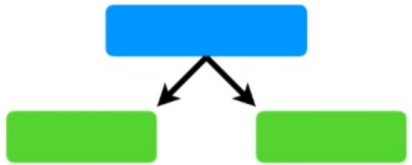
So we divide each **New Sample Weight** by **0.68** to get the normalized values.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

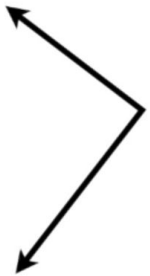
Now we just transfer the **Normalized Sample Weights** to the **Sample Weights** column, since those are what we will use for the next stump.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

Now we can use the modified **Sample Weights** to make the second **stump** in the forest.



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07



Alternatively, instead of using a **Weighted Gini Index**, we can make a new collection of samples that contains duplicate copies of the samples with the largest **Sample Weights**.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	
Yes	Yes	167	Yes	
No	Yes	156	No	
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

So we start by making a new, but empty, dataset that is the same size as the original…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

Then we pick a random number between 0 and 1…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

…and we see where that number falls when we use the **Sample Weights** like a distribution.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

If the number is between **0** and **0.07**, then we would put this sample into the new collection of samples…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

…and if the number is between **0.07** and **0.14** (**0.07** + **0.07** = **0.14**), then we would put this sample into the new collection of samples…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

…and if the number is between **0.14** and **0.21** (**0.14** + **0.07** = **0.21**), then we would put this sample into the new collection of samples…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

…and if the number is between **0.21** and **0.70** (**0.21** + **0.49** = **0.70**), then we would put this sample into the new collection of samples…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

For example, imagine the first number I picked was **0.72**…

…then I would put this sample into my new collection of samples…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No

Then I pick another random number and get **0.42**…

…and I would put this sample into my new collection of samples…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No
Yes	Yes	167	Yes
No	Yes	125	No

Then I pick **0.51**…

…and I would put this sample into my new collection of samples…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.07
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No
Yes	Yes	167	Yes
No	Yes	125	No
Yes	Yes	167	Yes

NOTE: This is the second time that we have added this particular sample to the new collection of samples.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes				
Yes				
No				
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

We then continue to pick random numbers and add samples to the new collection until we the new collection is the same size as the original.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No
Yes	Yes	167	Yes
No	Yes	125	No
Yes	Yes	167	Yes
Yes	Yes	167	Yes
Yes	Yes	172	No
Yes	Yes	205	Yes
Yes	Yes	167	Yes

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125		
Yes	No	168		
Yes	Yes	172		

Ultimately, this sample was added to the new collection of samples **4** times, reflecting its larger **Sample Weight**.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No
Yes	Yes	167	Yes
No	Yes	125	No
Yes	Yes	167	Yes
Yes	Yes	167	Yes
Yes	Yes	172	No
Yes	Yes	205	Yes
Yes	Yes	167	Yes

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	205	Yes	1/8
Yes	Yes	167	Yes	1/8

Lastly, we give all the samples equal **Sample Weights**, just like before.

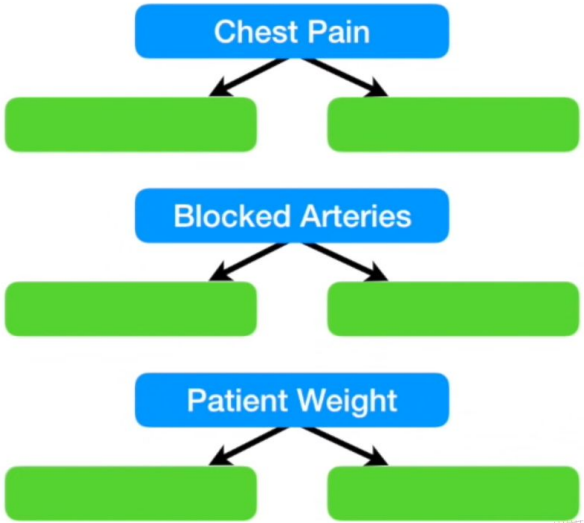
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	205	Yes	1/8
Yes	Yes	167	Yes	1/8

However, that doesn't mean the next stump will not emphasize the need to correctly classify these samples.

Because these samples are all the same, they will be treated as a block, creating a large penalty for being misclassified.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	205	Yes	1/8
Yes	Yes	167	Yes	1/8

Now we go back to the beginning and try to find the stump that does the best job classifying the new collection of samples.

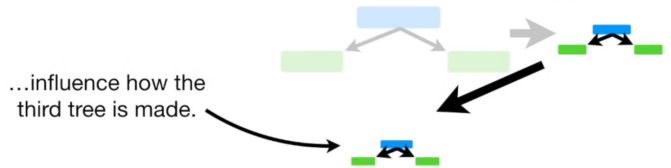


So that is how the errors that the first tree makes…

…influence how the second tree is made…

…and the errors that the second tree makes…

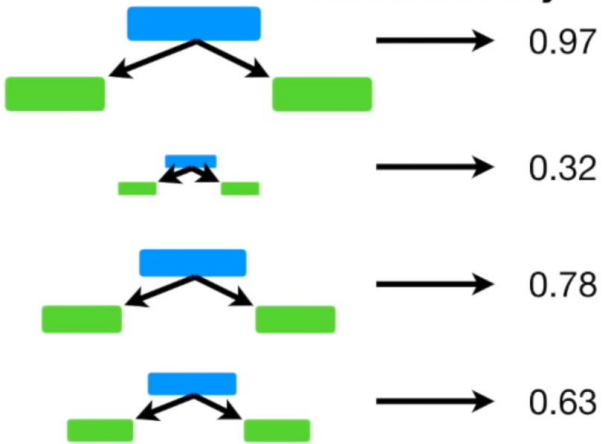
…influence how the third tree is made.



These are the **Amounts of Say** for these stumps…

Has Heart Disease

Amount of Say



Now we add up the **Amounts of Say** for this group of stumps…

Has Heart Disease **Total = 2.7**

…and these are the **Amounts of Say** for these stumps…

Does Not Have Heart Disease

Amount of Say

0.41

0.82

…and for this group of stumps…

Total = 1.23

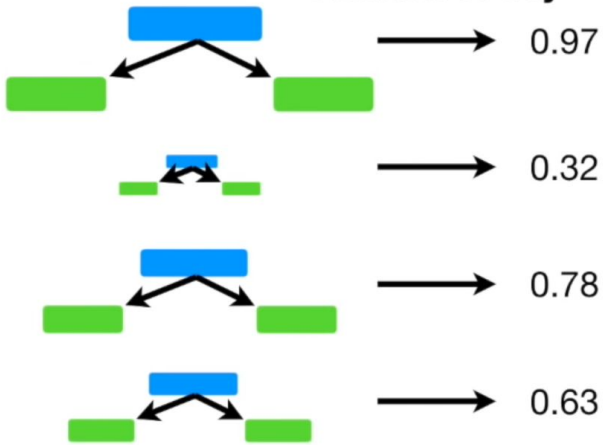
Does Not Have Heart Disease

Ultimately, the patient is classified as **Has Heart Disease** because this is the larger sum.

Has Heart Disease

Total = 2.7

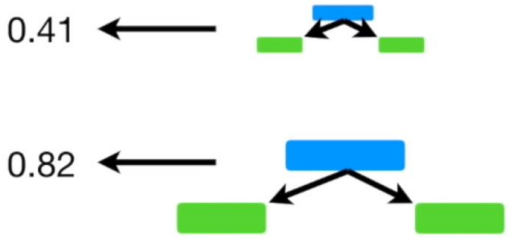
Amount of Say



Total = 1.23

Does Not Have Heart Disease

Amount of Say



Why AdaBoost?

Random Forests are also very good! They are probably close in average accuracy to boosted decision trees, but are less sensitive to outliers and parameter choices.

But everytime adaboost is not used because it takes greater training time. So problems that can be equally solved by Random forests should be solved by it.

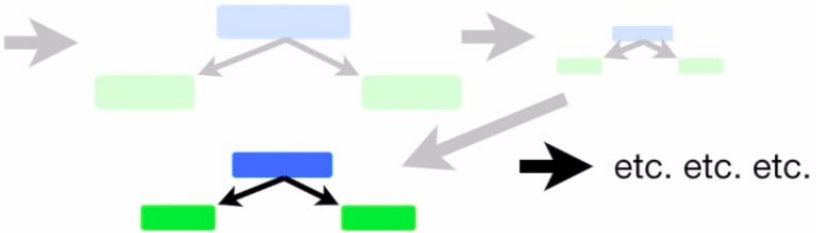
Gradient Boosted Decision Trees

Moving forward from Adaboost to Gradient Boosting

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

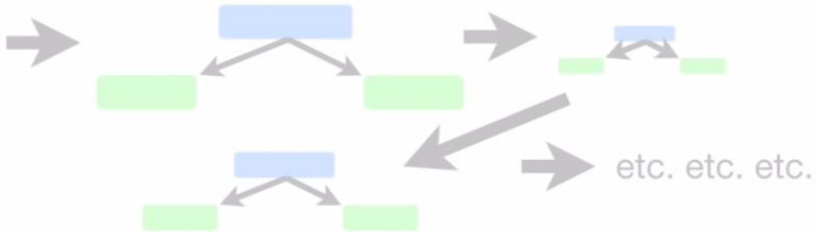
NOTE: When **Gradient Boost** is used to **Predict** a continuous value, like **Weight**, we say that we are using **Gradient Boost** for **Regression**.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
etc…	etc…	etc…	etc…



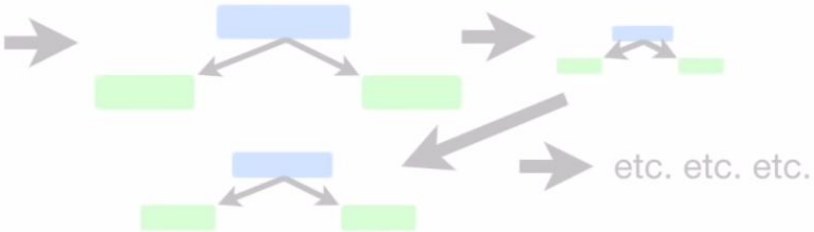
Then **AdaBoost** continues to make stumps in this fashion until it has made the number of stumps you asked for, or it has a perfect fit.

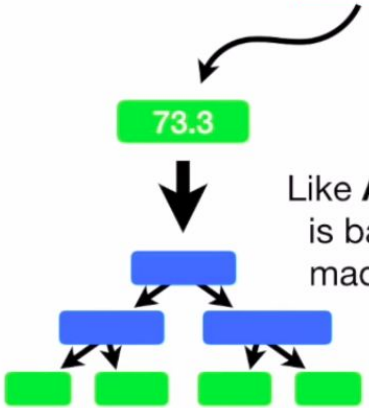
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
etc…	etc…	etc…	etc…



This leaf represents an initial guess for the **Weights** of all of the samples.

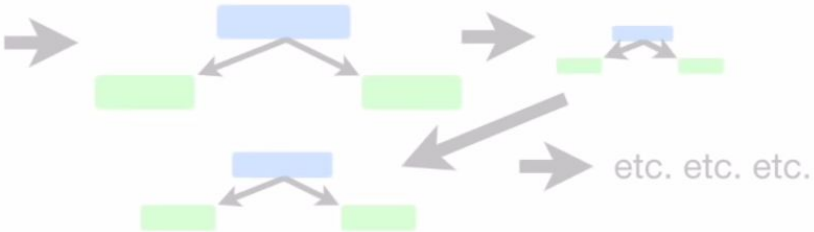
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
etc…	etc…	etc…	etc…

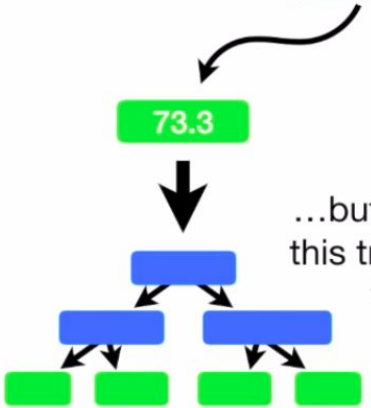




Like **AdaBoost**, this tree is based on the errors made by the previous tree…

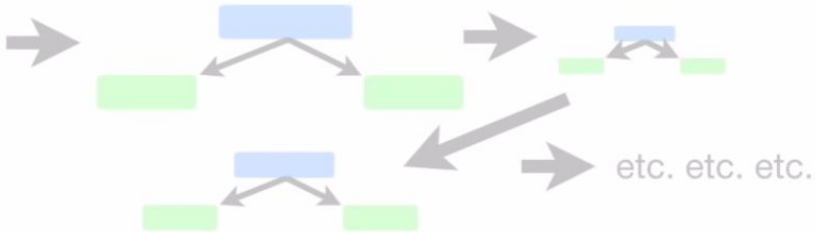
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
etc…	etc…	etc…	etc…



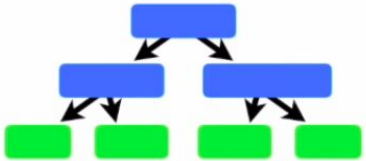


…but unlike **AdaBoost**, this tree is usually larger than a stump.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
etc…	etc…	etc…	etc…

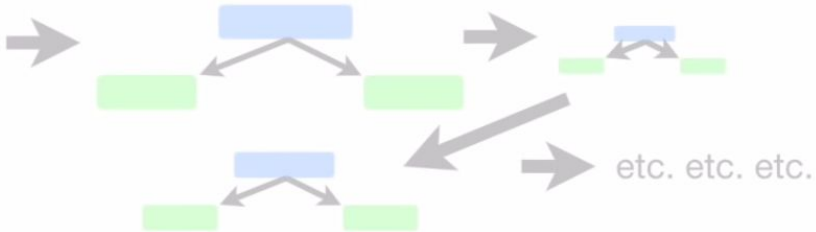


73.3

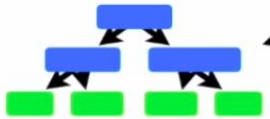


Thus, like **AdaBoost**, **Gradient Boost** builds fixed sized trees based on the previous tree's errors, but unlike **AdaBoost**, each tree can be larger than a stump.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
etc…	etc…	etc…	etc…

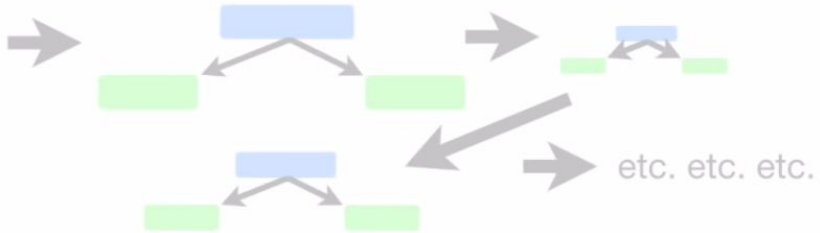


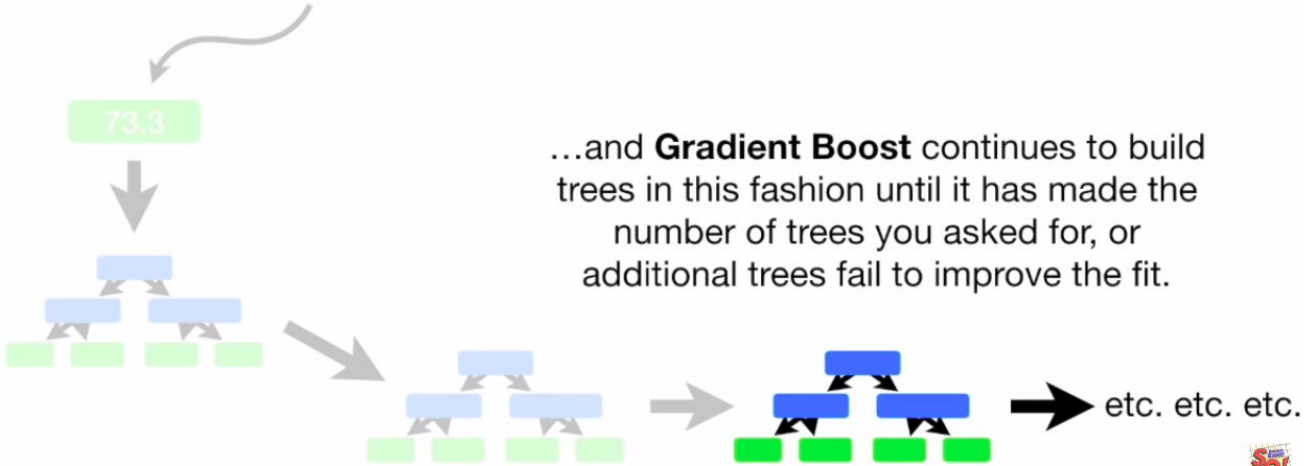
73.3



Also like **AdaBoost**, **Gradient Boost** scales the trees. However, **Gradient Boost** scales all trees by the same amount.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
etc…	etc…	etc…	etc…





…and **Gradient Boost** continues to build trees in this fashion until it has made the number of trees you asked for, or additional trees fail to improve the fit.

Average Weight

71.2

This is the first attempt at predicting everyone's weight.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

Average Weight

71.2

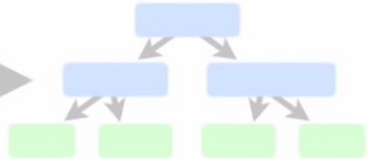
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

In other words, if we stopped right now, we would predict that everyone **Weighed 71.2** kg.

However, **Gradient Boost** doesn't stop here.

Average Weight

71.2

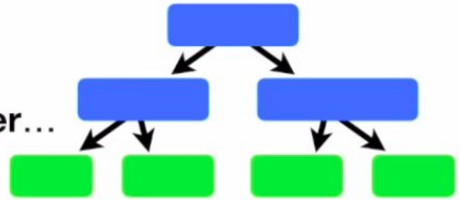


The errors that the previous tree made are the differences between the **Observed Weights** and the **Predicted Weight**, **71.2**.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

(**Observed Weight - Predicted Weight**)

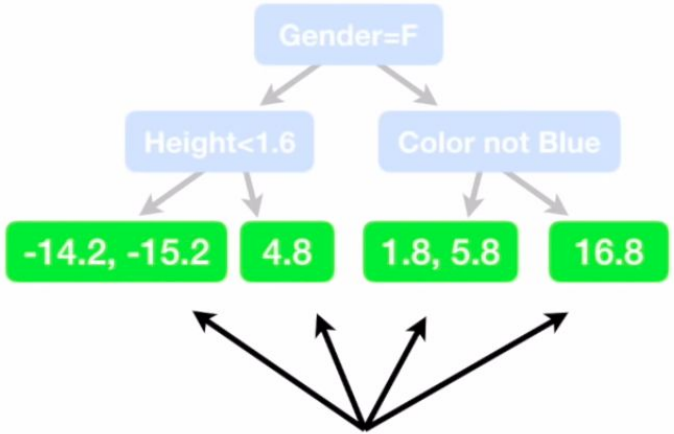
Now we will build a **Tree**, using **Height**, **Favorite Color** and **Gender**…



Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

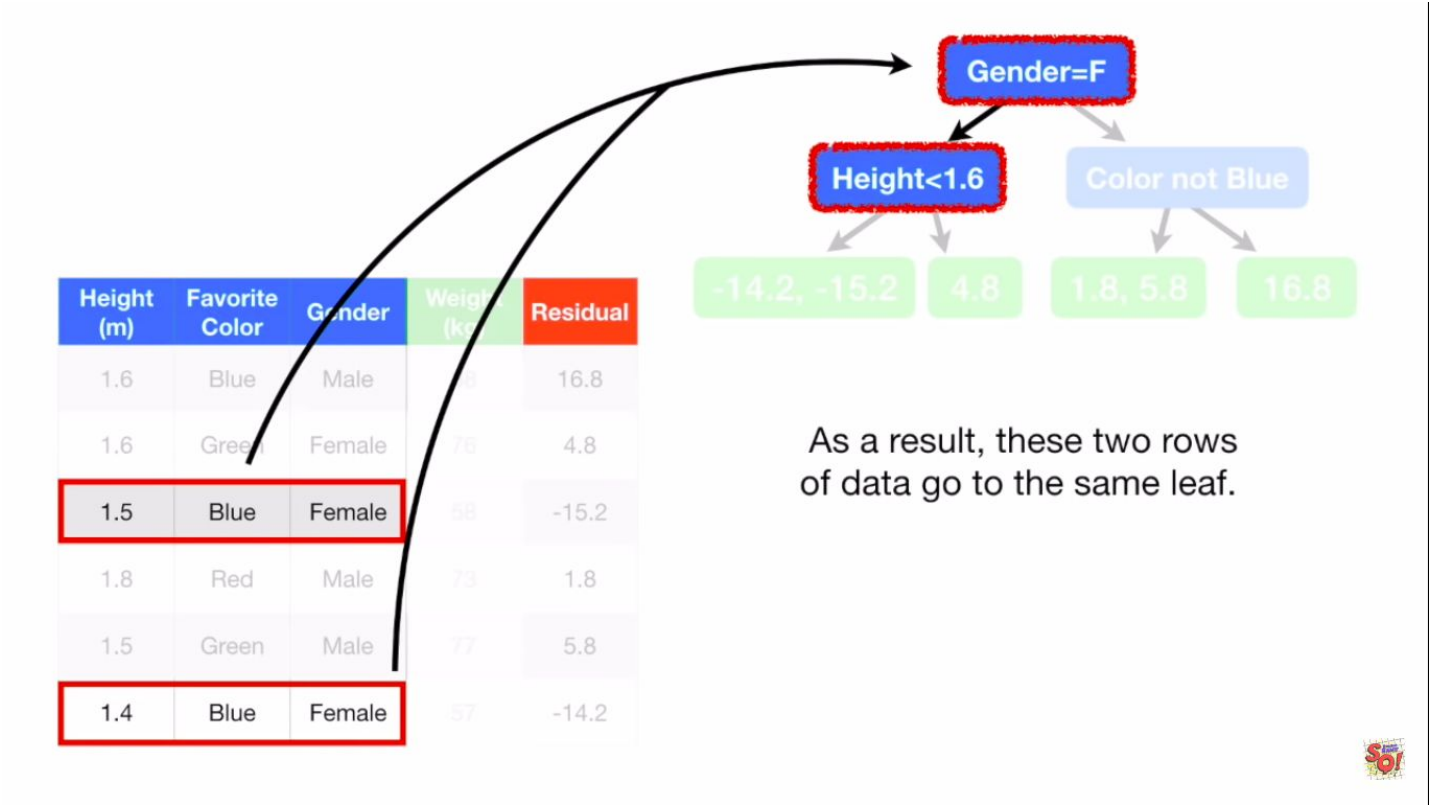
…to **Predict** the **Residuals**.

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

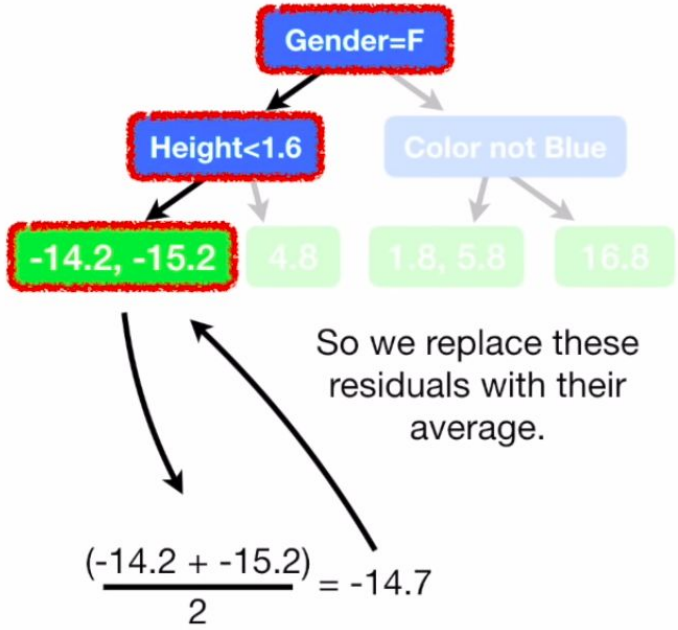


Remember, in this example we are only allowing up to four leaves…

…but when using a larger dataset, it is common to allow anywhere from **8** to **32**.


Gender=F
Height<1.6
Color not Blue
-14.2, -15.2
4.8
1.8, 5.8
16.8
Height (m)
Favorite Color
Gender
Residual
1.6
Blue
Male
16.8
1.6
Green
Female
4.8
1.5
Blue
Female
-15.2
1.8
Red
Male
1.8
1.5
Green
Male
5.8
1.4
Blue
Female
-14.2
As a result, these two rows of data go to the same leaf.

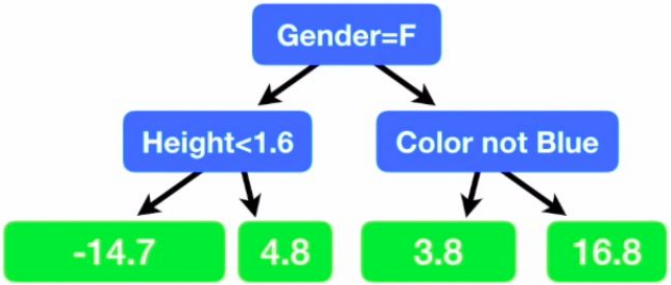
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2



Average Weight

71.2

+



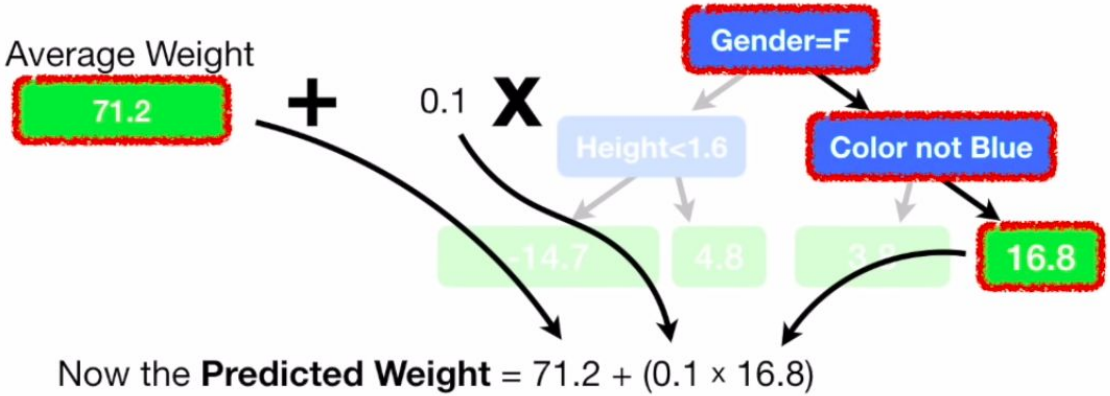
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

…to make a new **Prediction** of an individual's **Weight** from the **Training Data**.

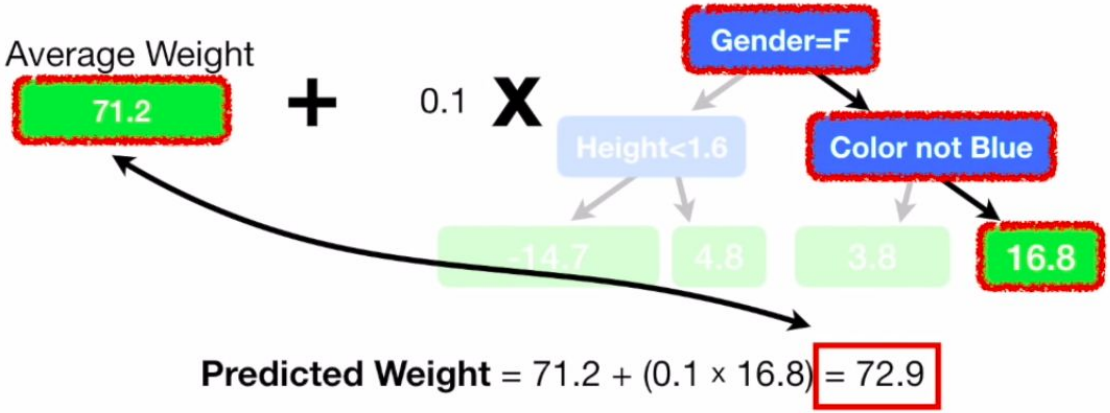
Predicted Weight = 71.2 + 16.8 = 88

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

In other words, we have low **Bias**, but probably very high **Variance**.

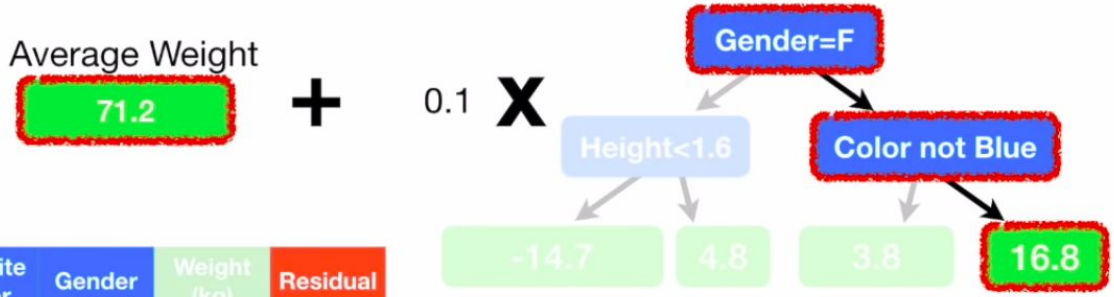


Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88



Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

…but it's a little bit better than the **Prediction** made with just the original leaf, which predicted that all samples would weigh **71.2**.

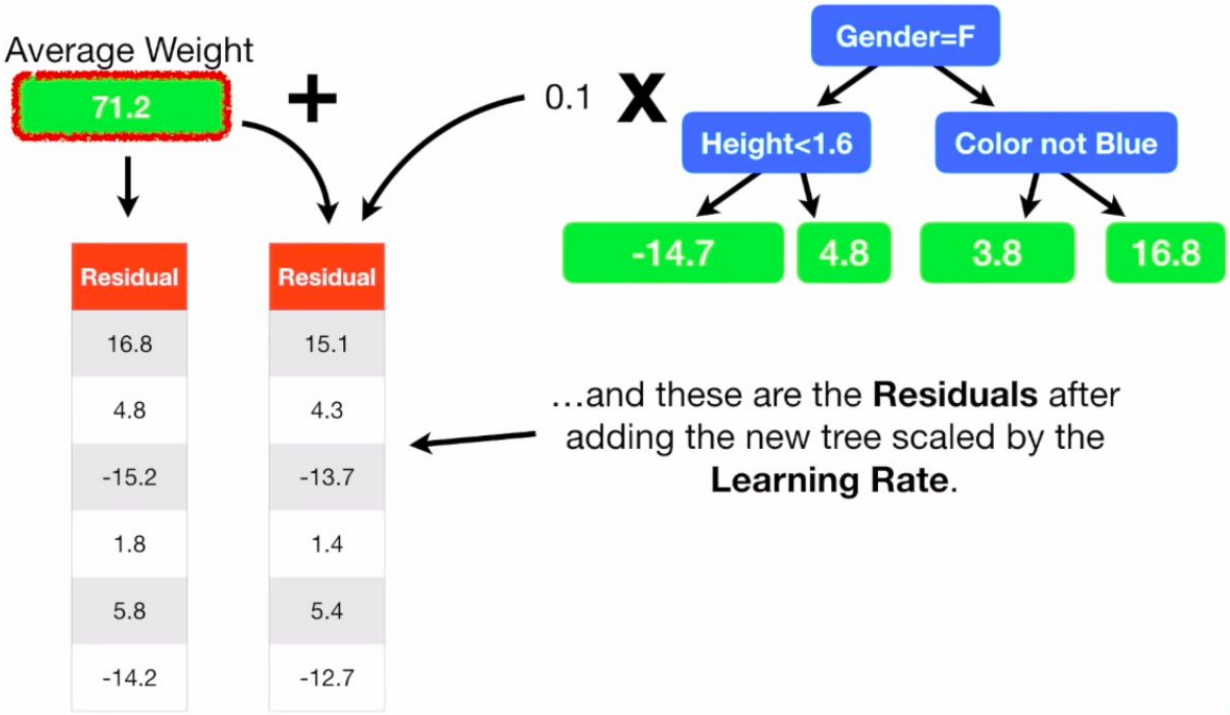


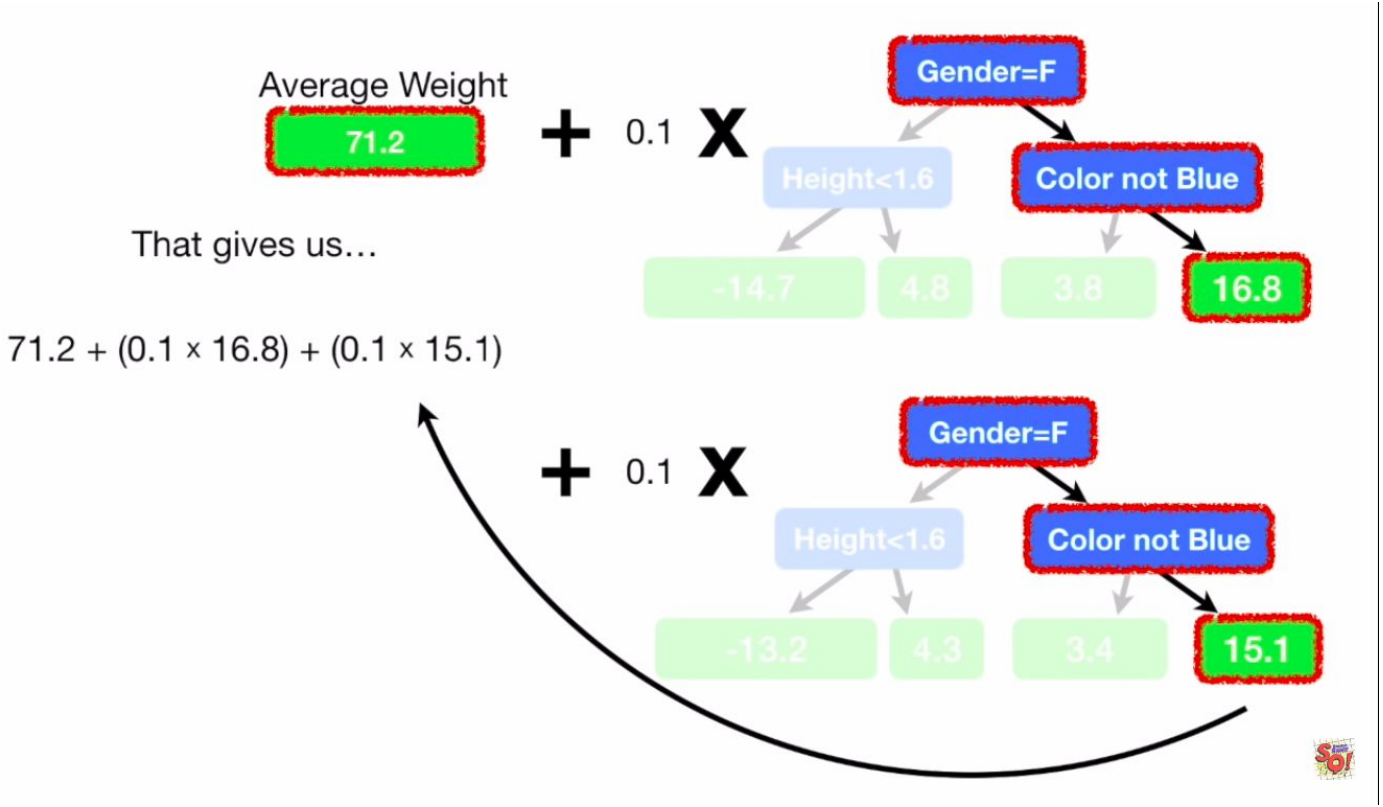
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	
1.5	Blue	Female	56	
1.8	Red	Male	73	
1.5	Green	Male	77	
1.4	Blue	Female	57	

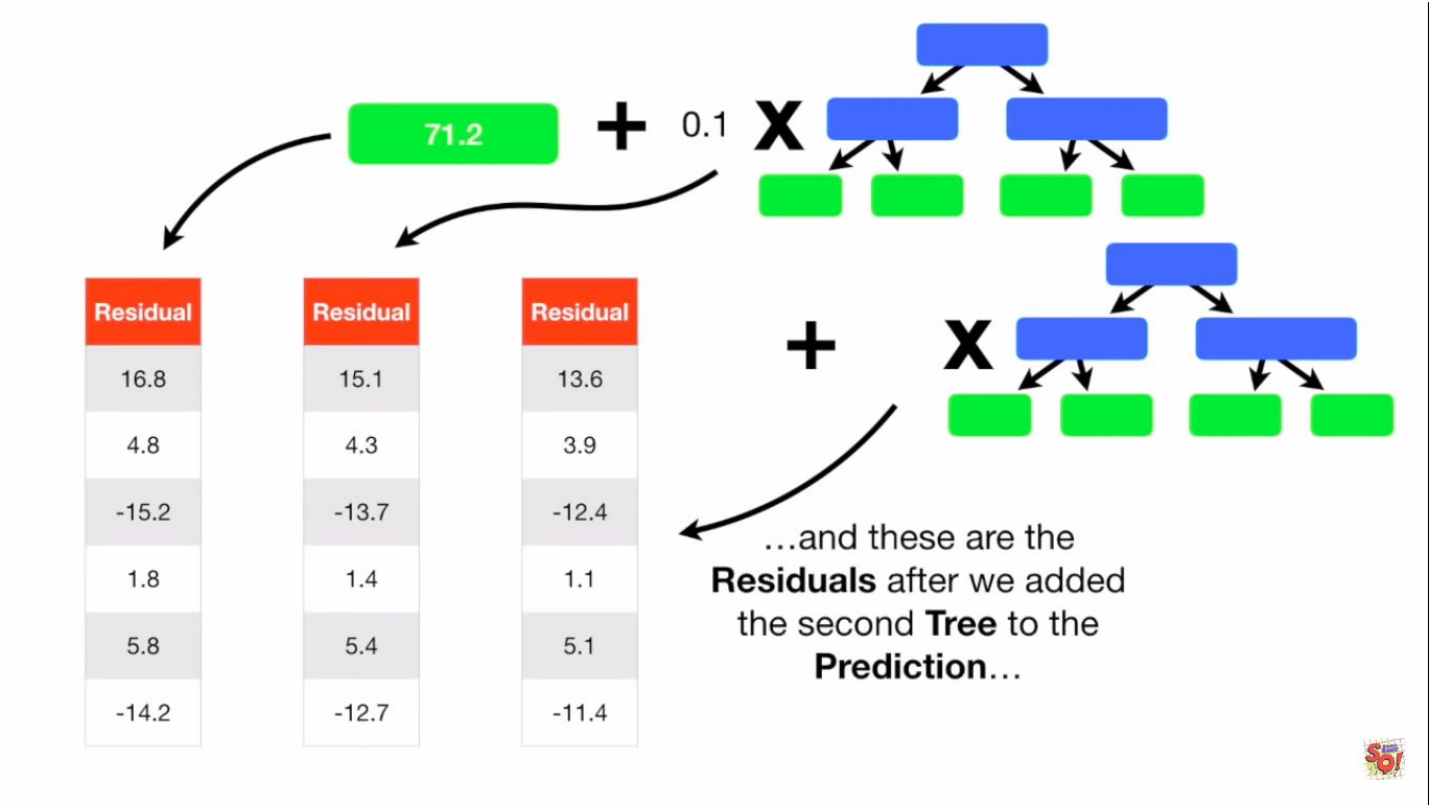
Residual = (**88** - (71.2 + 0.1 x 16.8))

= 15.1

…and we save that in the column for **Pseudo Residuals**.


Average Weight
71.2
+
0.1
X
Gender=F
Height<1.6
Color not Blue
-14.7
4.8
3.8
16.8
Residual
16.8
4.8
-15.2
1.8
5.8
-14.2
Residual
15.1
4.3
-13.7
1.4
5.4
-12.7
…and these are the **Residuals** after adding the new tree scaled by the **Learning Rate**.


Average Weight
71.2
+
0.1
X
Gender=F
Height<1.6
Color not Blue
-14.7
4.8
3.8
16.8
That gives us…
71.2 + (0.1 × 16.8) + (0.1 × 15.1)
+
0.1
X
Gender=F
Height<1.6
Color not Blue
-13.2
4.3
3.4
15.1


71.2
+
0.1
X
+
X
Residual
16.8
4.8
-15.2
1.8
5.8
-14.2
Residual
15.1
4.3
-13.7
1.4
5.4
-12.7
Residual
13.6
3.9
-12.4
1.1
5.1
-11.4
…and these are the **Residuals** after we added the second **Tree** to the **Prediction**…

Gradient Boosting for Classification

…and whether or not they love the movie **Troll 2**…

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

…then the **log(odds)** that someone **Loves Troll 2** is…

$$\log(\frac{4}{2}) = 0.7$$

log(4/2) = 0.7

Just like with **Logistic Regression**, the easiest way to use the **log(odds)** for **Classification** is to convert it to a **Probability**…

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

…and we do that with a **Logistic Function**.

$$\text{Probability of Loving Troll 2} = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$

log(4/2) = 0.7

Probability of **Loving Troll 2** = 0.7

NOTE: These two numbers, the **log(4/2)** and the **Probability** are the same only because I'm rounding. If I allowed **4** digits passed the decimal place…

$$\log(\frac{4}{2}) = 0.6931$$

$$\frac{e^{\log(4/2)}}{1 + e^{\log(4/2)}} = 0.6667$$

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

log(4/2) = 0.7

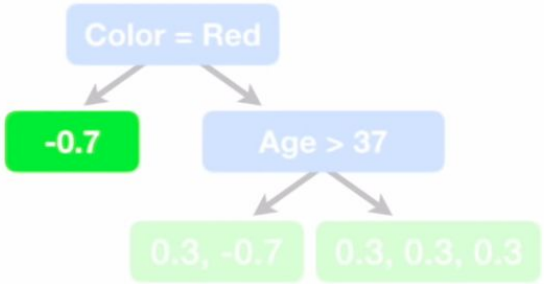
Probability of **Loving Troll 2** = 0.7

Likes Popcorn	Age	Favorite Color	Loves Troll 2	Residual
Yes	12	Blue	Yes	0.3
Yes	87	Green	Yes	0.3
No	44	Blue	No	-0.7
Yes	19	Red	No	-0.7
No	32	Green	Yes	0.3
No	14	Blue	Yes	0.3

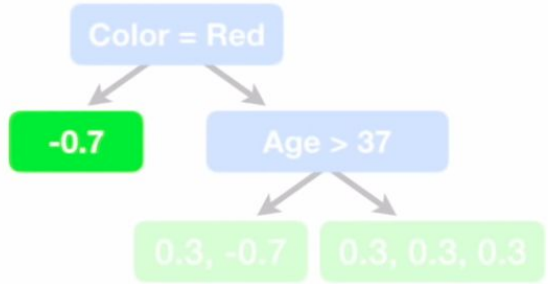
Hooray! We've calculated the **Residuals** for the leaf's initial **Prediction**.

log(4/2) = 0.7

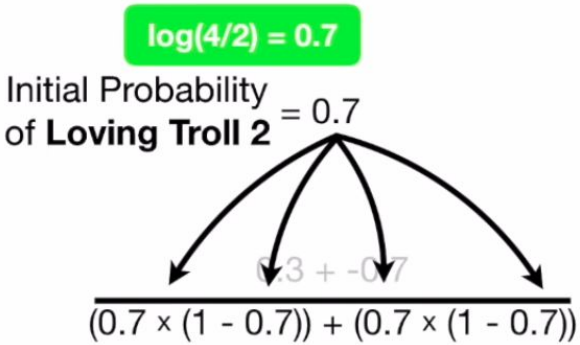
…so we can't just add them together to get a new **log(odds) Prediction** without some sort of transformation.

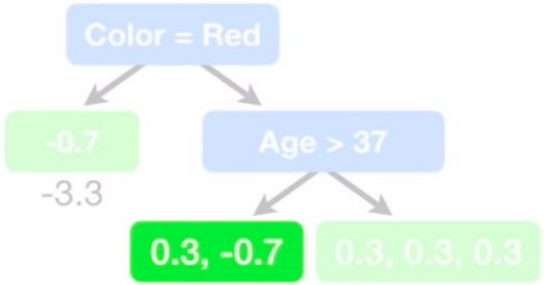


So for now, let's just use the formula to calculate the **Output Value** for this leaf.

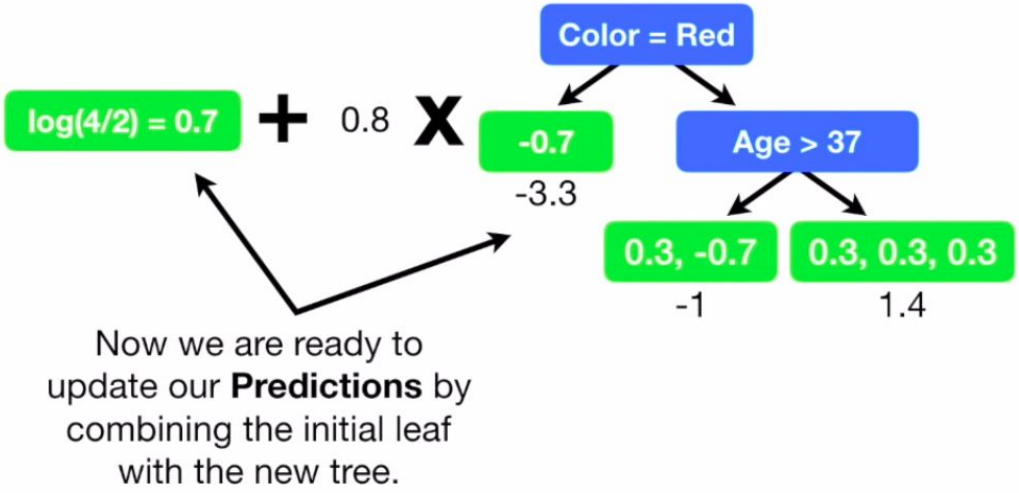


$$\frac{\sum \text{Residual}_i}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right]}$$





NOTE: For now, the **Previous Probabilities** are the same for all of the **Residuals**, but this will change when we build the next tree.


log(4/2) = 0.7
+
0.8
X
Color = Red
-0.7
-3.3
Age > 37
0.3, -0.7
-1
0.3, 0.3, 0.3
1.4
Now we are ready to update our **Predictions** by combining the initial leaf with the new tree.

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

…and the new **log(odds) Prediction** = **1.8**.

log(odds) Prediction = 0.7 + (0.8 × 1.4) = 1.8

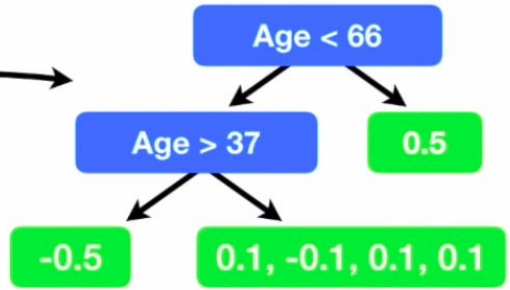
Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

…and the new
Predicted Probability = **0.9**…

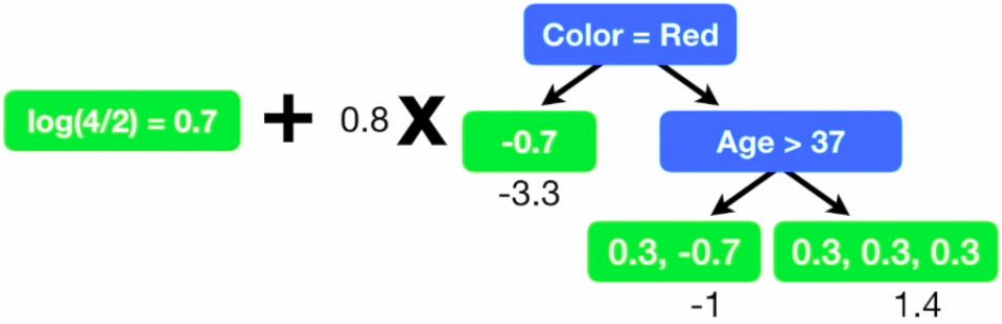
$$\text{Probability} = \frac{e^{1.8}}{1 + e^{1.8}} = 0.9$$

log(odds) Prediction = 0.7 + (0.8 × 1.4) = 1.8

Now that we have the **Residuals**, we can build a new tree…



Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.	Residual
Yes	12	Blue	Yes	0.9	0.1
Yes	87	Green	Yes	0.5	0.5
No	44	Blue	No	0.5	-0.5
Yes	19	Red	No	0.1	-0.1
No	32	Green	Yes	0.9	0.1
No	14	Blue	Yes	0.9	0.1



…and we scaled the this new tree with the **Learning Rate** as well.